

DSP Driver Software for Performance-Oriented Music Synthesis Systems

Adrian Freed and Keith Gordon
Center for New Music and Audio Technologies
Department of Music
1750 Arch Street
Berkeley, CA 94709
adrian@cnmat.berkeley.edu

ABSTRACT: We describe a software interface between high level musical performance environments and low level software drivers for digital signal multiprocessors. We describe the first implementation of a driver using the proposed interface for low-cost, commercially available, DSP56001-based signal processors (Reson8™, SoundAccelerator™, AudioMedia™) and the way the driver is used from within the HyperCard and MAX environments.

Introduction

Current performance-oriented music synthesis systems consist of interconnected gestural devices, general purpose workstations and special purpose synthesis hardware.

Software running on the general purpose workstation handles musical abstractions, note event and control stream data, and the user interface. Special purpose processors handle computational requirements of music synthesis algorithms more efficiently than general purpose workstation processors. MIDI, a loosely coupled and relatively slow protocol has been used to connect these processing elements. However, there is a trend towards more tightly coupled, higher speed bus interfaces [Lowe 1989] and more signal processing nodes [Barrière et al 1989, Wawrzyniek 1989,90].

The price/performance ratio of workstations and digital signal processors is improving at a furious pace. In contrast, software to make this hardware musically useful takes a long time to develop and mature. The primary goal of the driver interface described here is to hide signal processing device specific details from musical applications. By implementing the interface in portable ANSI C, we also expect to be able to transfer it to new workstations without difficulty.

We have interfaced the driver to HyperCard, MAX [Puckette 1988], and MacMix [Freed 1987]. Future target environments include SmallTalk, Formula [Anderson 1986], HMSL [Rosenboom 1985], and Lisp. Each of these have a strong following and something unique to offer in musical applications.

Driver Functionality

Conventional operating system driver interfaces cannot be used in performance-oriented musical applications because they do not address the need to accurately express when events in devices should occur. Drivers implementing the interface we propose are responsible for timely exchange of synthesis and analysis parameters between host and multiprocessor nodes.

Parameters are described symbolically in the domain of a signal processing algorithm. A driver maps these names to addresses of processing nodes it is responsible for, converts parameters between the natural number representations of the host and signal processors, transmits parameters to processor nodes and finally synchronizes update of these parameters to real-time clocks.

Control parameters are represented as vectors of floating point numbers. This choice was based partly on the nature of the computations in the control structures for synthesis algorithms we are interested in. The overriding consideration, however, was the need for a portable, well defined representation which could be mapped efficiently to the representation used in both fixed point and floating point signal processors. Industry backing of the IEEE floating point standard and decreasing incremental system cost of floating point vindicate this choice.

The driver also performs housekeeping jobs for the signal processors such as identifying, testing and loading them with code from descriptions stored in the workstation.

Driver for Motorola DSP56001

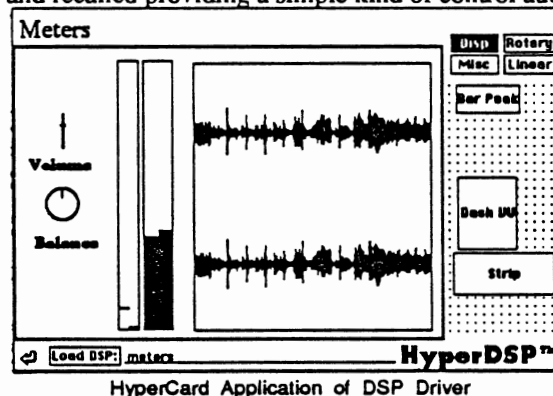
The first implementation of a driver using the interface is for low-cost, commercially available, DSP56001-based [Motorola 1989] signal processors: Reson8™, SoundAccelerator™, AudioMedia™, MacMix Excellerator™. This driver has been interfaced to HyperCard, MacMix and MAX.

The driver scans cards installed in a Macintosh computer and fills a configuration table with information on cards with DSP resources. It matches the resources available to those required by compiled algorithms stored in files of DSP56001 machine code. It also extracts the names and properties of parameters for these algorithms from a symbol table.

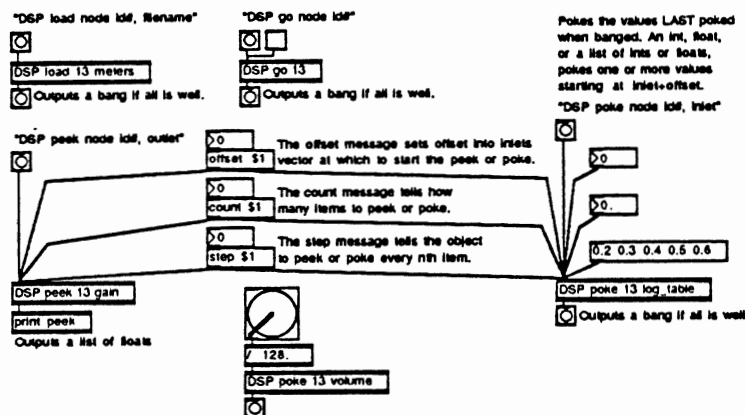
The ability to perform timely, atomic updates of control parameters is essential to many types of signal processing. We are using the tuning of a bank of one hundred 2-pole resonators as a test case. Not only do more than 300 coefficients need to be loaded from the workstation to the signal processor, but also these coefficients are quite expensive to compute from a high level description in the workstation environment. We are experimenting with the well-known double buffering technique for timely delivery of updates. DSP programs execute using parameters from one buffer while another buffer is filled by the host workstation. When a buffer of updates is complete and it is time for the update to occur, the buffers are switched.

Environments Interfaced to Driver

The driver is used in the HyperDsp environment—a HyperCard-based system for rapid prototyping of test harnesses for DSP code. Users can build control panels and displays interactively from a palette of standard parts: linear and rotary faders, strip charts and bar displays. These parts are then wired symbolically to the signal processing code. Buttons are provided to load each node in a DSP network individually. Midi controller messages can also be associated with the on-screen faders for more satisfying gestural control. The ability to store a different front panel on each card of a stack makes it very convenient to collect related algorithms. For example, there is a stack of cards which run diagnostics code to test DSP systems element by element. These test cards are arranged in the stack in a natural order for debugging hardware. The state of controls on each card is automatically stored and recalled providing a simple kind of control automation.



The DSP driver has been encapsulated into an object for the MAX environment [Puckette 1988] using its external code resource facility. This object responds to messages to configure and load code into DSP nodes and read and write vectors of parameters. MAX has proven to be a good host environment for the DSP driver. Its strong and efficient real-time scheduling, MIDI support, and table lookup allow musicians to quickly integrate new DSP algorithms into their performances. MAX is also very useful for the rapid prototyping and simulation essential to developing new DSP algorithms for sound synthesis and processing.



Max Object Encapsulation of DSP Driver

Conclusion

Inevitably the driver interface will have to be refined to suit new workstation and signal processing architectures. However, it is a solid base on which to build environments for music performance which can take advantage of the exciting and rapid developments in music software, workstations and VLSI signal processing.

References

- Anderson D., Kuivila R., *Accurately Timed Generation of Discrete Musical Events*, Computer Music Journal 10(3), pp. 48-56.
- Jean-Baptiste Barrière, Adrian Freed, Pierre-François Baisnée, Marie-Dominique Baudot, (1989), *A Digital Signal Multiprocessor and its Musical Application*, International Computer Music Conference 1989.
- Freed A. (1987), *Recording, Mixing, and Signal Processing on a Personal Computer*, Proceedings of the AES 5th International conference on Music and Digital Technology, pp. 158 -162
- Lowe W., Currie R., (1989), *Digidesign's Sound Accelerator: Lessons Lived and Learned* Computer Music Journal, 13(1) pp. 36-46
- Motorola, (1989), *DSP56000/DSP56001 Digital Signal Processor User's Manual*, Motorola Literature Distribution, PO Box 20912, Phoenix, Arizona 85036
- Potard Y., Baisnée P-F., Barrière J-B., (1986), *Experimenting with Models of Resonance Produced by a New Technique for the Analysis of Impulsive Sounds*, Proceedings of 1986 International Computer Music Conference, La Haye, Computer Music Association, pp.269-274
- Puckette M. (1988), *The Patcher*, Proceedings of the 14th International Computer Music Conference, Köln, 1988, Feedback Studio Verlag, available from Computer Music Association.
- Rosenboom D., Polansky, L., (1985), *HMSL (Hierarchical Music Specification Language): a real-time environment for formal, perceptual and compositional experimentation*, Proceedings ICMC, San Francisco, Computer Music Association.
- Wawrzynek J. C., von Eicken T., (1989), *Mimic, a custom VLSI Parallel Processor for Musical Sound Synthesis*, Proceedings of IFIP VLSI 89, Munich, FRG.
- Wawrzynek J. C., von Eicken T., (1990), *VLSI Parallel Processing for Musical Sound Synthesis*, Proceedings of ICMC, Glasgow.